アルゴリズム 58－6
（1997. 9. 19）

組合せ論理回路に対するイベント駆動による再評価

ビャトキン・バレリー 中野浩嗣 林達也
名古屋工業大学電気情報工学科

本論文では、回路で表された論理関数の再評価を行う効率良いアルゴリズムを提案する。アルゴリズムは前処理と再評価の２つの部分から構成される。論理回路と初期入力が与えられたときに、前処理では再評価のためのデータ構造を構成する。再評価では変更のあった入力を受け取り、論理回路の出力を更新する。前処理の計算時間は回路の大きさの線形時間であり、再評価の時間は、変更のあった入力の数の線形時間である。

Event-driven evaluation of combinatorial logic circuits

Valery Viatkin, Koji Nakano, Tatsuya Hayashi

Dept.of Electrical and Computer Engineering
Nagoya Institute of Technology
Showa-ku, Nagoya 466,Japan
{valery,nakano,hayashi}@elcom.nitech.ac.jp
tel./fax.:+81-52-735-5450

This paper presents an efficient algorithm for re-computation of a Boolean function represented as an acyclic and non-reduced logic circuit. The algorithm consists of pre-computation and re-computation parts. For a given logic circuit and initial inputs, the pre-computation constructs data structure for the re-compuation. The re-computation accepts a list of triggered inputs and updates the result of the logic circut. Computing time of the pre-computation is linear to the size of the circuit and of the re-computation is linear to the number of triggered inputs.

1 Introduction

Speed of logic computations is important for many real-time computer applications. Especially it is true for logic control systems where responce charateristic is critical. Distributed nature of many controlled objects urges to redirect attention to event-oriented computations since explicit data aquisition and update of all output variables sometimes takes in such systems unacceptable time proportional to the total number of inputs at best.

Principle of event-driven computation, studied in [7], [11] offers a selective approach improving responce on account of starting computations without delay, choice only the functions actually required to be updated, and use of more efficient recomputation of each function comparing to the case when it is computed completely.

State of a real-time system is completely defined by current value X of input vector $\mathbf{X} \in \{0,1\}^N$. *Event* σ is simultaneous triggering of subset $\mathbf{X}_{[\sigma]} \in \mathbf{X}$ that changes value of $\mathbf{X}$ to X^σ. The event can be explicitly defined by list $\sigma =< j_1, j_2, ..., j_k >$ of indices of triggered inputs. It is also assumed that $k << N$. The task of event-driven recomputation of a Boolean function $F : \mathbf{X} \to \{0,1\}$ asks to find its value $F(X^\sigma)$ given $F(X)$, *event* and some intermediate data precomputed in advance.

Efficiency of the recomputation as well as of the full computation, is strongly dependent on the presentation of formula. Many models of logic computation are based on use of graph presentations. Most popular are binary decision diagrams (BDD) [2], ordinary traverse of which provides the result in time linear to the number of inputs N. BDD traverse checks a corresponding variable at every node and choses one of two alternative directions to continue. Event-driven computation of correspondigly modified BDDs proposed in [9, 10] recomputes a function in a time range $O(k) - O(klogN)$ depending on the memory and amount of required precomputations. However, application of the BDDs is of limited practical use for their exponential size.

The need of more efficient data structure made us to study conventional logic circuits in their most trivial case of $fan-in = 2$, $fan-out = 1$. The principle of event-driven traverse of the circuits is closer to the method of clause counter map, introduced by Welch in [7] and evolved then in [8]. Contribution of the present work is in finding of the solution for the general case i.e. at assumption that there is an arbitrary number of a variable entries into the formula and $(k \geq 1)$.

Let the function F be given initially as an expression in basis **{and, or, nand, nor}**. Even though every variable in general can be of an arbitrary number of entries into the formula, in practical cases assumption about linear number of each variable entries seems to be quite reasonable. Thus total number of affected entries k_e in case of an event of order k would be of $k_e = O(k)$ order.

Event-driven computation finds *difference* of a Boolean function instead of recomputing its value. The *difference* of a Boolean value v taken in discrete time is denoted as $\partial v(t) = v(t) \oplus v(t-1)$ and true iff the value changes. Correspondingly for function $F(X)$ at event σ difference is $\partial F(X) = F(X) \oplus F(X^\sigma)$ and $F(X^\sigma) = F(X) \oplus \partial F(X)$. To study transitional behaviour of Boolean functions (i.e. their dependence on triggering of a certain group of input variables) Boolean derivative $\frac{\partial F}{(\partial x_{j_1}, \partial x_{j_2}, ..., \partial x_{j_k})}$ of order k was introduced ([5], [6]). It has been proven that at event $\sigma = (j_1, j_2, \ldots, j_k)$ difference is equal to the derivative of order k: $\partial F(X) = \frac{\partial F}{(\partial x_{j_1}, \partial x_{j_2}, ..., \partial x_{j_k})}$. Using this result the following simple properties of basic logic operations were derived in [8]. Let $y(X) = f_1(X)\mathbf{op}f_2(X)$, where $\mathbf{op} \in \mathbf{\{and, nand, or, nor\}}$.

Proposition 1.1 *For* $\mathbf{op} \in \mathbf{\{and, nand\}}$: $\frac{\partial y}{\partial f_1} = f_2$, *while for* $\mathbf{op} \in \mathbf{\{or, nor\}}$: $\frac{\partial y}{\partial f_1} = \overline{f_2}$.

Proposition 1.2 *For all operations* $\mathbf{op} \in \mathbf{\{and, nand, or, nor\}}$: $\frac{\partial y}{(\partial f_1 \partial f_2)} = (f_1 \equiv f_2)$.

These properties enable us to substitute Boolean calculations for operations with differences in order to derive difference of the function by event propagation through a formula from variable entries to the result of the whole function. When the function is presented as a kind of a binary tree computation looks like back traverse of the tree from leaves to the root controlled by rules 1.1,1.2.

2 Event logic circuits

Logic circuits built of gates **{and, or, nand, nor}** is most easily constructed binary tree presentation of a Boolean formula. Every operation in the formula corresponds to the node in the circuit and every variable to a leaf. In order to represent circuit in homogeneous basis of gates it can be equaly presented by a circuit of the same topology in basis **{or, nor}** applying De Morgan laws to **{and, nand }** gates. Moreover, assuming that all gates have no more than one incoming edge, it can be presented as **or**-gates circuit with edges marked with negation attribute. And since rules 1.1,1.2 prove that negation doesn't affect dependence of operation difference on differences of operands, the attributes can be omitted to produce event logic circuit (or ELC for short) which is used further as a basic data structure for difference recomputation purposes.

Example 1 *Consider Boolean function:* $f = \overline{(\overline{x_2}x_3)}x_5 \vee x_1x_4 \vee x_2 \vee \overline{(x_2 \vee x_4)}\,\overline{x_4}\,\overline{(\overline{x_2} \vee x_5 \vee x_1)}$. *Composition of equivalent event logic circuit is presented in figure 1.*

Thus, we define logic circuit (LC) in the following way. It is a binary tree $C = \langle V, \mathbf{l}, \mathbf{r}, v_0, \psi \rangle$, with V standing for the base set of nodes, also denoted as $V = \mathcal{V}(C)$, $v_0 \in V$ - root. Children of node v are $\mathbf{l}(v), \mathbf{r}(v)$ and the parent is $\mathbf{p}(v)$. Subcircuit rooted in w is termed $\lceil w \rceil$. Set V divides onto two subsets: set V_L of terminal nodes or leaves and set V_G of functional nodes or gates: $V = V_G \cup V_L$; $V_G \cap V_L = \emptyset$. Assume that all gates of the circuit are **or** - gates and the circuit is *non-reduced* i.e. every nodes except terminal ones have no more than one parent (i.e. fan-in=1). In this case attribute of inversion can be moved from edges to the nodes: $neg : V \to \{0,1\}$ though it will be shown as a dot on the incoming edge. Each terminal node is associated with a logic variable by mapping $\psi : V_L \to \mathbf{X}$. Circuit $\lceil v \rceil$ defines a Boolean function recursively as $F_v = F_{\mathbf{l}(v)}\mathbf{or}F_{\mathbf{r}(v)}) \oplus neg(v)$ if v is a functional node, or as $F_v = \psi(v) \oplus neg(v)$, if it is a terminal node. Value of the function F_v, stored as a *functional marking* of node: $v.F$, helps to complete full circuit evaluation in as many as $O(|V|)$ steps.

As in any tree, in an LC exists a single path $[w, v_0]$ from node w to the root. Length of the path is termed

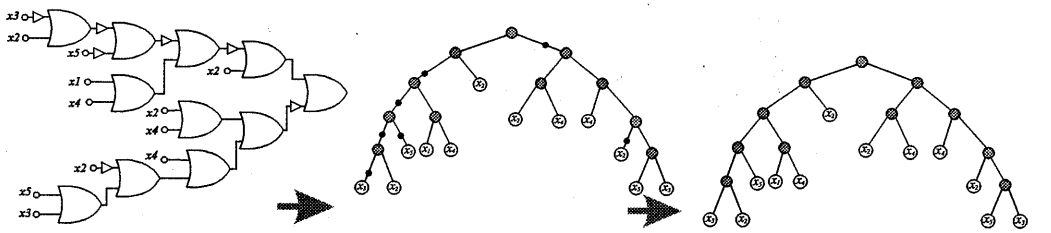

Figure 1: Construction of event logic circuit from OR-NOT circuit

as $rank(w)$; $\max_{w \in V}(rank(w))$ termed as a rank of an LC. The relation of vicinity $\eta : V \backslash v_0 \to V \backslash v_0$ uniquely maps the sibling node for every node except the root in the following way: $\eta(v) = \mathbf{r}(\mathbf{p}(v)$ if $v = \mathbf{l}(\mathbf{p}(v))$, or $\eta(v) = \mathbf{l}(\mathbf{p}(v))$ if $v = \mathbf{r}(\mathbf{p}(v))$. Difference $\partial(v)$ of node v is difference of the Boolean function F_v determined by LC $\lceil v \rceil$.

The *event-driven recomputation* problem of a Boolean formula can be formulated in terms of ELCs as follows. Given a non-reduced ELC with functional marking in every node corresponding to the state X and a query, defined as a subset of terminal nodes where functional marking is changed value at state X^σ, the problem asks to update the functional marking in the root of the LC. The set V_L^σ is called as a *set of event nodes*. We assume that its size k_e is linearly dependent on the the event dimension, i.e.: $O(k_e) = O(k)$. For a sub-circuit $\lceil v \rceil$ set of its event nodes is termed as $\mathcal{V}(\lceil v \rceil)_L^\sigma$. Term class of sub-LCs such that $\left|\mathcal{V}(\lceil v \rceil)_L^\sigma\right| = i$ as $\mathbb{E}^i$. Solution of this problem is done in the following steps: first we consider single-event case $\mathbb{E}^1$ and two-event case $\mathbb{E}^2$. The latter is used as a basic means to solve the general case $\mathbb{E}^{>2}$ by cancelling it to several $\mathbb{E}^2$ sub-circuits.

Assume that before the re-computation begins only event nodes have differences equal to 1. Conditions of propositions 1.1,1.2 can be formulated in terms of logic circuits as follows. Predicate meaning that change of function in node v is not blocked by the value of functional marking in its child $w \in \{\mathbf{l}(v), \mathbf{r}(v)\}$ (Prop. 1.1) is: $\alpha(v, w) = (F_w = 0)$. Predicate determining whether functional markings in both successors of the node are equal (Prop. 1.2) is termed $\varepsilon(v) = (F_{\mathbf{l}(v)} = F_{\mathbf{r}(v)})$.

3 Properties of event logic circuits

3.1 General properties of event nodes

The facts stated in this chapter provide a basement for the generic algorithm of reductions in LCAs set that eventually leads to the finding of difference of the function.

Term two nodes $v_1, v_2 \in V$ as *independent* $v_1 \sim v_2$ if $(v_1 \notin \lceil v_2 \rceil)\&(v_2 \notin \lceil v_1 \rceil)$. Thus all nodes in V_L are mutually independent, and any set of event nodes, as far as it is subset of V_L, also holds this property. For a pair of independent nodes $a \sim b$, there exist at least one node c, such that $a \in \lceil c \rceil$ and $b \in \lceil c \rceil$. Such a node with highest rank is called *lowest common ancestor* (LCA) of the two nodes. That of a group W of more than two mutually independent nodes can be defined as a node $\mathcal{L}(W)$ such that $\lceil \mathcal{L}(W) \rceil$ consists of all nodes of W, whereas none of $\lceil \mathbf{l}(\mathcal{L}(W)) \rceil$ or $\lceil \mathbf{r}(\mathcal{L}(W)) \rceil$ does.

From now on assume the set of event nodes $\mathbf{N}_\sigma = V_L^\sigma$ is sorted in ascending order. Relation of strict order can be defined on the set of independent nodes as follows:

Definition 1 $a < b \Leftrightarrow a \in \lceil \mathbf{l}(\mathcal{L}(a,b)) \rceil$ *and* $b \in \lceil \mathbf{l}(\mathcal{L}(a,b)) \rceil$ *(Fig.2,a)*

Subcircuit rooted in $\mathcal{L}(\mathbf{N}_\sigma)$ is a minimal one dependent on all the changed variables. The following theorem provides LCA's major place in the event-oriented algorithm of difference computation.

Theorem 3.1 *Difference of a LC equals to difference of* $\mathcal{L}(\mathbf{N}_\sigma)$.

The theorem implies a way of a difference computation - first to find LCA of set $\mathbf{N}_\sigma$ and difference in it given the set of event nodes and marking of ELC. To do that we use the following properties of LCA for series of 2-3 event nodes.

Lemma 3.2 $(b \in \lceil a \rceil)\&(c \in \lceil b \rceil) \Rightarrow (c \in \lceil a \rceil)$.

Lemma 3.3 $a < b < c \Rightarrow \lceil \mathcal{L}(a,b) \rceil \subseteq \lceil \mathcal{L}(a,c) \rceil$.

Proof: Suppose that the opposite is true, i.e.: $\exists a < b < c \Rightarrow \lceil \mathcal{L}(a,b) \rceil \supset \lceil \mathcal{L}(a,c) \rceil$. It means that $c \in \lceil \mathcal{L}(a,b) \rceil$. Since $c > b \Rightarrow \exists d = \mathcal{L}(b,c) : c \in \lceil \mathbf{r}(d) \rceil$, and as $b > a \Rightarrow b \in \lceil \mathbf{r}(\mathcal{L}(a,b)) \rceil$ (Fig.2,b). Set of $\lceil \mathcal{L}(a,b) \rceil$ nodes consists of $\mathcal{L}(a,b) \cup \mathcal{V}\lceil \mathbf{r}(\mathcal{L}(a,b)) \rceil \cup \mathcal{V}\lceil \mathbf{l}(\mathcal{L}(a,b)) \rceil$. By the definition, $(c \in \lceil \mathbf{r}(\mathcal{L}(a,b)) \rceil)\&(c \notin \lceil \mathbf{l}(\mathcal{L}(a,b)) \rceil)$, so it can be concluded that $\mathcal{L}(a,c) = \mathcal{L}(a,b)$, since $\lceil \mathbf{l}(\mathcal{L}(a,b)) \rceil$ does not contain c, $\lceil \mathbf{r}(\mathcal{L}(a,b)) \rceil$ does not contain a, but $\mathcal{L}(a,b)$ contains both a and c. That does not agree with the assumption made. ■

Lemma 3.4 $\forall a < b < c \in V \Rightarrow \lceil \mathcal{L}(a,b) \rceil \subset \lceil \mathcal{L}(b,c) \rceil \vee \lceil \mathcal{L}(b,c) \rceil \subset \lceil \mathcal{L}(a,b) \rceil$.

Proof: Statement of the previous lemma is equivalent to: $\forall a < b < c \in V \Rightarrow \lceil \mathcal{L}(a,b) \rceil = \lceil \mathcal{L}(a,c) \rceil \vee \lceil \mathcal{L}(a,b) \rceil \subset \lceil \mathcal{L}(a,c) \rceil$.

If $\lceil \mathcal{L}(a,b) \rceil = \lceil \mathcal{L}(a,c) \rceil$ then $\lceil \mathcal{L}(b,c) \rceil \subset \lceil \mathcal{L}(a,b) \rceil$, since $\mathcal{L}(b,c) \in [b, \mathbf{r}(\mathcal{L}(a,b))] \Rightarrow \lceil \mathcal{L}(b,c) \rceil \subseteq \lceil \mathbf{r}(\mathcal{L}(a,b)) \rceil \Rightarrow \lceil \mathcal{L}(b,c) \rceil \subset \lceil \mathcal{L}(a,b) \rceil$.

If $\lceil \mathcal{L}(a,b) \rceil \subset \lceil \mathcal{L}(a,c) \rceil$ then, first, $\lceil \mathcal{L}(a,b) \rceil \subseteq \lceil \mathbf{l}(\mathcal{L}(a,c)) \rceil$, since otherwise it would be true that $(\lceil \mathcal{L}(a,b) \rceil \subseteq \lceil \mathbf{r}(\mathcal{L}(a,c)) \rceil)\&(c \in \lceil \mathbf{r}(\mathcal{L}(a,c)) \rceil)$, and consequently $\lceil \mathbf{l}(\mathcal{L}(a,c)) \rceil$ does not contain any of a, b, c, implying that $\mathcal{L}(a,c)$ is not an LCA for a, c. Secondly $c \in \lceil \mathbf{r}(\mathcal{L}(a,c)) \rceil$, and therefore $\mathcal{L}(a,c) = \mathcal{L}(b,c)$, i.e. $\lceil \mathcal{L}(b,c) \rceil \subset \lceil \mathcal{L}(a,b) \rceil$ ■

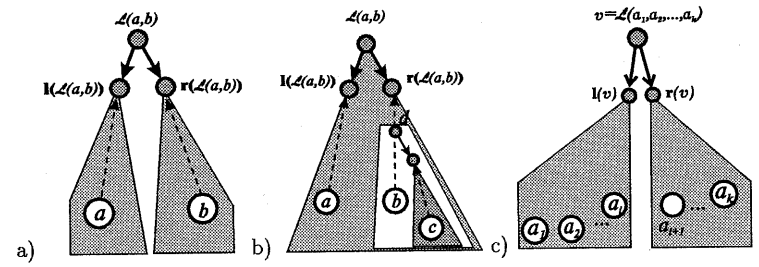

Figure 2: Properties of LCA

Lemma 3.5 *For an arbitrary triple of independent nodes $a < b < c$ it is valid that: $\forall d \in \lceil b \rceil (a < d < c)$.*

Proof: Prove $a < d$. Since $(d \in \lceil b \rceil)\&(b \in \mathbf{r}(\mathcal{L}(a,b))) \Rightarrow d \in \mathbf{r}(\mathcal{L}(a,b))$. It is obvious also that $\mathcal{L}(a,b) = \mathcal{L}(a,d)$, implying $a < d$. Similarly can be proved $d < c$ ■

The set of event $\mathbf{N}_\sigma$ nodes can be regarded as an ordered set of independent nodes. Thus evaluation of difference can be reduced to evaluation of difference of this set, which, in turn, to the case of $\mathbb{E}^2$ circuits by structurizing the set of pairwise LCAs of event nodes.

Theorem 3.6 *In an arbitrary ordered set of independent nodes $\mathbf{A} = a_1 < a_2 < \ldots < a_k$ there exist a unique pair $a_i, a_{i+1} \in \mathbf{A} : \mathcal{L}(a_i, a_{i+1}) = \mathcal{L}(\mathbf{A})$.*

Proof:1) Existence: Since $\lceil \mathcal{L}(\mathbf{A}) \rceil = \{\mathcal{L}(\mathbf{A})\} \cup \mathcal{V}\lceil \mathbf{r}(\mathcal{L}(\mathbf{A})) \rceil \cup \mathcal{V}\lceil \mathbf{l}(\mathcal{L}(\mathbf{A})) \rceil$ and $\mathcal{L}(\mathbf{A}) \neq a_i (\forall i \leq k)$ (i.e. otherwise set of nodes would not be independent) then $\mathbf{A} \in \mathcal{V}\lceil \mathbf{r}(\mathcal{L}(\mathbf{A})) \rceil \cup \mathcal{V}\lceil \mathbf{l}(\mathcal{L}(\mathbf{A})) \rceil$. By the definition of LCA $\exists i : 1 \leq i \leq k : \{a_1, \ldots, a_i\} \in \lceil \mathbf{l}(\mathcal{L}(\mathbf{A})) \rceil$ and $\{a_{i+1}, \ldots, a_k\} \in \lceil \mathbf{r}(\mathcal{L}(\mathbf{A})) \rceil$(Fig. 2,c). Thus (a_i, a_{i+1}) is the pair sought for. Really, $\mathcal{L}(a_i, a_{i+1}) = \mathcal{L}(\mathbf{A})$ since $a_i \notin \lceil \mathbf{r}(\mathcal{L}(\mathbf{A})) \rceil, a_{i+1} \notin \lceil \mathbf{l}(\mathcal{L}(\mathbf{A})) \rceil$, whereas both $a_i, a_{i+1} \in \mathcal{L}(\mathbf{A})$.

2) Uniqueness: In the ordered sequence $\mathbf{A} = a_1 < a_2 < \ldots < a_k$ the pair $(a_i, a_{i+1}) : \mathcal{L}(a_i, a_{i+1}) = \mathcal{L}(\mathbf{A})$ is unique, as far as any other pair a_j, a_{j+1} at $j < i$ belongs to $\lceil \mathbf{l}(\mathcal{L}(\mathbf{A})) \rceil$, and therefore $\lceil \mathcal{L}(a_j, a_{j+1}) \rceil \subset \lceil \mathcal{L}(\mathbf{A}) \rceil$ ■

Let $\mathfrak{M}_A = \{\mathcal{L}(a_i, a_{i+1}) | i \leq k-1, a_i \in \mathbf{A}\}$ be the set of LCAs for pairs of nodes with consequent indices. This set contains $k-1$ elements and, according to the latter theorem, includes $\mathcal{L}(\mathbf{A})$.

Node v is called *reacheable by a simple path* from node $w : w \to v$, if $w \in \mathfrak{M}_A$, $v \in \mathfrak{M}_A \cup \mathbf{A}$, $\exists [w, v] : \forall s \in]w, v[\Rightarrow s \notin \mathfrak{M}_A$, i.e simple path does not include any nodes of $\mathfrak{M}_A$ except first and last ones.

Minimal double-event ELC is an ELC, root of which is an LCA of its two event nodes. Let $\mathbb{ME}^2$ be designation of the class of such circuits. The following property of $\mathbb{ME}^2$ circuits is valid:

Proposition 3.7 *In the minimal double-event LC $\lceil c \rceil$ with event nodes a, b the root is reacheable from both a and b by simple paths: $c \to a, c \to b$.*

The following theorem proves that in the set $\mathfrak{M}_A$ for every node v there are exactly two nodes, from which it is reacheable by a simple path.

Theorem 3.8 *In an arbitrary ordered set of independent event nodes $\mathbf{A} = a_1 < a_2 < \ldots < a_k, (k > 1)$ it is valid that $\forall m \in \mathfrak{M}_A (\exists! m_1, m_2 \in \mathfrak{M}_A \cup \mathbf{A} : (m_1 \to m)\&(m_2 \to m))$.*

3.2 $\mathbb{ME}^2$ circuits

Since $\mathbb{ME}^2$ circuit is explicitly defined by a pair of two event nodes, denote it as a triple $\langle v, e_1, e_2 \rangle$, where $v = \mathcal{L}(e_1, e_2)$. The following property of $\mathbb{ME}^2$ directly follows from the definition of LCA.

Lemma 3.9 $\lceil w \rceil \in \mathbb{ME}^2 \Rightarrow \lceil \mathbf{l}(w) \rceil \in \mathbb{E}^1$ *and* $\lceil \mathbf{r}(w) \rceil \in \mathbb{E}^1$.

Consider an $\mathbb{E}^1$ sub-circuit with event node v. According to the proposition 1.1, difference of its parent is equal to 1 only if $\alpha(\mathbf{p}(v), \eta(v)) = 1$. The same is valid for the parent of $\mathbf{p}(v)$ and so on up to the root. Let ancestors of v form the list $a_1 = \mathbf{p}(v), a_2 = \mathbf{p}(a_1) = \mathbf{p}(\mathbf{p}(v)), \ldots, a_{rank(v)} = v_0$. Node a_i at $i < rank(v)$ and $\partial(a_i) = 1, \partial(a_{i+1}) = 0$ or just $\partial(a_i) = 1$ at $i = rank(w)$ is termed as $\mathcal{D}(v)$ - maximal dependent ancestor (MDA for short) of node v. Note that all nodes of the path $v, a_1, a_2, \ldots, \mathcal{D}(v)$ have the same MDA.

Add the following descriptors to each node of ELC: $v.mda$ stores MDA, $v.\alpha_l = \alpha(v, l(v))$, $v.\alpha_r = \alpha(v, r(v))$ and $v.\varepsilon = \varepsilon(v)$. Precomputation of MDA in all nodes of ELC can be done in $O(|V|) = O(N)$ time.

For $C = \langle v, w_1, w_2 \rangle \in \mathbb{ME}^2$ the following opportunities exist:

1. MDA of the both nodes include children of the root $\mathcal{L}(w_1, w_2)$;
2. MDA of either w_1 or w_2 includes a child of $\mathcal{L}(w_1, w_2)$;
3. MDA of none nodes includes $\mathcal{L}(w_1, w_2)$.

ALGORITHM 1

```
function TRIG(w, w1, w2:node): Boolean;
//w = L(w1, w2)
begin
case
  R(w1, l(w)) and R(w2, r(w)):  return(w.ε);
  R(w1, l(w) and not R(w2, r(w)):  return(w.αr);
  not R(w1, l(w) and R(w2, r(w)):  return(w.αl);
  else return(0)
end;end.
```

To express relation between MDA and LCAs denote predicate $\mathfrak{R}(w, v) = \lceil \mathcal{D}(w) \rceil \supseteq \lceil v \rceil$. Then, if $\mathfrak{R}(w_1, v)$ then $\partial(\mathbf{l}(v)) = 1$, if $\mathfrak{R}(w_2, v)$ then $\partial(\mathbf{r}(v)) = 1$. So in case (1) proposition 1.2 works, giving $\partial(v) = v.\varepsilon$, in case (2) proposition 1.1 gives $\partial(v) = v.\alpha_l$ or

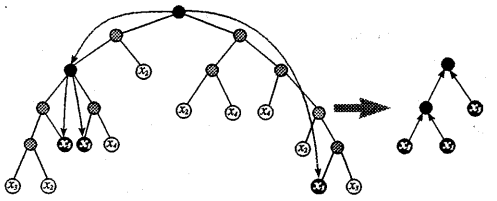

Figure 3: Signal diagram of event $\sigma = \langle 1, 5 \rangle$

$\partial(v) = v.\alpha_r$ and in case (3) $\partial(v) = 0$, since $(\partial(\mathbf{l}(v)) = 0)$ and $(\partial(\mathbf{r}(v)) = 0)$. Thereby we've proven corectness of algorithm $TRIG(w, w_1, w_2{:}\text{node})$, where $w = \mathcal{L}(w_1, w_2)$, computing difference for two given nodes.

3.3 Signal diagrams

Using the mapping "reachability by a simple path" $\pi : \mathfrak{M}_A \cup \mathbf{A} \to \mathfrak{M}_A$ it is possible to construct a binary tree with nodes from the set $\mathfrak{M}_A \cup \mathbf{A} \subset V$ and edges corresponding to the mapping π - nodes that belong to $\pi^{-1}(v)$ become children of node v in the tree. The total number of nodes is $|\mathfrak{M}_A| = 2k - 1 = O(k)$. The tree is refered to as a *signal diagram* (SD) $S = \langle \mathfrak{M}_A, \mathbf{A}, \pi \rangle$ at fixed ELC and given event σ.

According to theorem 3.1 difference in the root of signal diagram is equal to difference of the initial ELC, so that interpretation of SD to find difference is a way to find result of the function. Since size of SD is of $O(k)$ order rough estimation of the complexity of the interpretation also appears to be of $O(k)$ order, that is quite attractive.

For example, outline signal diagram for event $\sigma = \langle 1, 5 \rangle$ in ELC considered above. There 3 event nodes in the ELC and resulting SD is shown in figure 3. If the difference had been found in $O(3)$ worktime, it would be quite different from time of the total formula computation proportional to the size of whole ELC that is 23 nodes.

The algorithm described further finds the difference in the root of SD given the set of event nodes in $O(k)$ time. It builds and processes signal diagram corresponding to the event.

4 Bottom-up interpretation of event logic circuits

4.1 Removal of pairs of minimal LCA

Therefore, the sought algorithm has the following objectives: given a set of event nodes where differences are equal to 1, it to find difference in the root. Basic step of the algorithm lies in finding of currently minimal pair and removal it from $\mathbf{N}_\sigma$. Let on this step current signal diagram be $S = \langle \mathfrak{M}_{\mathbf{N}_\sigma}, \mathbf{N}_\sigma, \pi \rangle$. There is at least one node in $\mathfrak{M}_{\mathbf{N}_\sigma}$ such that both its children belong to $\mathbf{N}_\sigma$. Term the subset of all such nodes as $min(\mathfrak{M}_{\mathbf{N}_\sigma})$. Let $v \in min(\mathfrak{M}_{\mathbf{N}_\sigma}) : v = \mathcal{L}(w_1, w_2); w_1, w_2 \in \mathbf{N}_\sigma$. Since $\lceil v \rceil \in \mathbb{ME}^2$ difference in it can be found directly applying $TRIG$.

If the difference is 1 that means v can be considered as an event node instead of w_1, w_2 which to be excluded from $\mathbf{N}_\sigma$. This procedure is called further as *reduction* and lies in *cancelling* of C to $C' = C \setminus \lceil v \rceil$ and signal diagram S to $S' = \langle \mathfrak{M}_{\mathbf{N}_\sigma} \setminus v, \mathbf{N}_\sigma \setminus w_1, w_2 \cup v, \pi \rangle$. Thus the total number of nodes in the SD decreases. Otherwise, if the difference in v is 0, then the whole subtree rooted in v and containing event nodes w_1, w_2 as well can be excluded out of consideration since events in w_1, w_2 do not propagate upper than v. In this case $C' = C - \lceil v \rceil$, $S' = \langle \mathfrak{M}_{\mathbf{N}_\sigma} \setminus v, \mathbf{N}_\sigma \setminus w_1, w_2, \pi \rangle$ and number of nodes in the SD also decreases. Since SD shrinks at every reduction, applying the above step until it consists only of the root we eventually derive difference of the function. The following criteria provides a way to find a pair with mininal LCA in $\mathfrak{M}_{\mathbf{N}_\sigma}$ in order to expose it to the reduction.

Theorem 4.1 (Criteria of minimality) *1. At* $j = 1 (\mathcal{L}(a_j, a_{j+1}) \in \min(\mathfrak{M}_{\mathbf{N}_\sigma})) \Leftrightarrow \lceil \mathcal{L}(a_1, a_2) \rceil \subset \lceil \mathcal{L}(a_2, a_3) \rceil$ *(leftmost minimum);*

2. $\forall 1 < j < k - 2 (\mathcal{L}(a_j, a_{j+1}) \in \min(\mathfrak{M}_{\mathbf{N}_\sigma})) \Leftrightarrow \mathcal{L}(a_{j-1}, a_j) \supset \lceil \mathcal{L}(a_j, a_{j+1}) \rceil \subset \lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil$ *(intermediate minimum);*

3. At $j = k - 1 (\mathcal{L}(a_j, a_{j+1}) \in \min(\mathfrak{M}_{\mathbf{N}_\sigma})) \Leftrightarrow \lceil \mathcal{L}(a_j, a_{j+1}) \rceil \subset \lceil \mathcal{L}(a_{j-1}, a_j) \rceil$ *(rightmost minimum);*

Proof: The case (2) will be considered, as it is the most complex. Necessity $(\mathcal{L}(a_j, a_{j+1}) \in \min(\mathfrak{M}_{\mathbf{N}_\sigma})) \Rightarrow \lceil \mathcal{L}(a_{j-1}, a_j) \rceil \supset \lceil \mathcal{L}(a_j, a_{j+1}) \rceil \subset \lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil)$ follows from the fact that by lemma 3.4 there exist generally only two cases: either $\lceil \mathcal{L}(a_{j-1}, a_j) \rceil \subset \lceil \mathcal{L}(a_j, a_{j+1}) \rceil$ or $\lceil \mathcal{L}(a_{j-1}, a_j) \rceil \supset \lceil \mathcal{L}(a_j, a_{j+1}) \rceil$. But the first case is impossible since $\mathcal{L}(a_j, a_{j+1})$ is minimal, therefore, by the definition, it doesn't include any element of $\mathfrak{M}_{\mathbf{N}_\sigma}$. Applying similar reasoning to the pair $\mathcal{L}(a_j, a_{j+1}), \mathcal{L}(a_{j+1}, a_{j+2})$ we conclude that $\lceil \mathcal{L}(a_{j-1}, a_j) \rceil \supset \lceil \mathcal{L}(a_j, a_{j+1}) \rceil \subset \lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil$.

To prove sufficiency $\lceil \mathcal{L}(a_{j-1}, a_j) \rceil \supset \lceil \mathcal{L}(a_j, a_{j+1}) \rceil \subset \lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil \Rightarrow (\mathcal{L}(a_j, a_{j+1}) \in \min(\mathfrak{M}_{\mathbf{N}_\sigma}))$ assume that $\lceil \mathcal{L}(a_{j-1}, a_j) \rceil \supset \lceil \mathcal{L}(a_j, a_{j+1}) \rceil \subset \lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil$ is true, but there exist $(a_p, a_{p+1}) : \lceil \mathcal{L}(a_p, a_{p+1}) \rceil \subset \lceil \mathcal{L}(a_j, a_{j+1}) \rceil$. For instance consider the case $p > j$. As far as $\lceil \mathcal{L}(a_j, a_{j+1}) \rceil \subset \lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil$, then $a_{j+2} \notin \lceil \mathcal{L}(a_j, a_{j+1}) \rceil$. If $\lceil \mathcal{L}(a_p, a_{p+1}) \rceil \subset \lceil \mathcal{L}(a_j, a_{j+1}) \rceil$ then by lemma 3.3 $(a_p, a_{p+1} < a_{j+2}) \Rightarrow j < p < p+1 < j+2 \Rightarrow p = j+2$. But this is impossible since $\lceil \mathcal{L}(a_j, a_{j+1}) \rceil \subset \lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil$ ■

Reduction of a single pair doesn't disrupt non-reduced nature of ELC. Similarly, set $\mathbf{N}'_\sigma$ remains to be a set of independent nodes like $\mathbf{N}_\sigma$. The algorithm of reduction relies on the number of properties inherited to pairs of nodes satisfying to the criteria of minimality. Some of them are formulated in the following lemma:

Lemma 4.2 *If* $v = \mathcal{L}(a_j, a_{j+1}) : \lceil \mathcal{L}(a_{j-1}, a_j) \rceil \supset \lceil v \rceil \subset \lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil$ *then:*

1. *Either* $\lceil \mathcal{L}(a_{j-1}, a_j) \rceil \subset \lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil$ *or* $\lceil \mathcal{L}(a_{j-1}, a_j) \rceil \supset \lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil$;
2. *Both* $\mathcal{L}(a_{j+1}, v) = \mathcal{L}(a_{j-1}, a_j)$ *and* $\mathcal{L}(v, a_{j+2}) = \mathcal{L}(a_{j+1}, a_{j+2})$;
3. $(v \sim a_{j-1}), (v \sim a_{j+2})$ *and* $(a_{j-1} < v < a_{j+2})$;

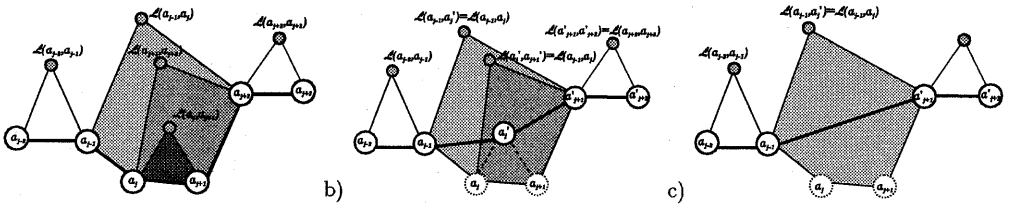

Figure 4: Rules of the removal

4. $\mathcal{L}(a_{j-1}, a_{j+2}) = \max(\mathcal{L}(a_{j-1}, a_j), \mathcal{L}(a_{j+1}, a_{j+2}))$;

Proof:

1. Since both the subtrees $\lceil \mathcal{L}(a_{j-1}, a_j) \rceil$, $\lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil$ have common subtree $\lceil \mathcal{L}(a_j, a_{j+1}) \rceil$, one of them must contain the other.

2. As $\lceil v \rceil \subset \lceil \mathcal{L}(a_{j-1}, a_j) \rceil \Rightarrow a_{j-1} \in \lceil \mathbf{l}(\mathcal{L}(a_{j-1}, a_j)) \rceil$ and $v \in \lceil \mathbf{r}(\mathcal{L}(a_{j-1}, a_j)) \rceil$. None of $\lceil \mathbf{l}(\mathcal{L}(a_{j-1}, a_j)) \rceil$, $\lceil \mathbf{r}(\mathcal{L}(a_{j-1}, a_j)) \rceil$ contains both a_{j-1} and v, hence $\mathcal{L}(a_{j-1}, v) = \mathcal{L}(a_{j-1}, a_j)$.

3. The fact $v \sim a_{j-1}$ and $v \sim a_{j+2}$ trivially follows from that $a_{j-1} \notin \lceil \mathcal{L}(a_j, a_{j+1}) \rceil$ and $a_{j+2} \notin \lceil \mathcal{L}(a_j, a_{j+1}) \rceil$ as $\lceil \mathcal{L}(a_{j-1}, a_j) \rceil \supset v \subset \lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil$. Let us prove that $a_{j-1} < v < a_{j+2}$. In (2) it is proven that $a_{j-1} \in \lceil \mathbf{l}(\mathcal{L}(a_{j-1}, a_j)) \rceil$ and $v \in \lceil \mathbf{r}(\mathcal{L}(a_{j-1}, a_j)) \rceil$, therefore $a_{j-1} < v$. Similarly $v < a_{j+2}$.

4. Assume for determinacy that $\lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil \subset \lceil \mathcal{L}(a_{j-1}, a_j) \rceil$. Then $a_{j-1} \in \lceil \mathcal{L}(a_{j-1}, a_j) \rceil$. Node $a_{j+2} \in \lceil \mathcal{L}(a_{j-1}, a_j) \rceil$, but $a_{j+2} \notin \lceil \mathbf{l}(\mathcal{L}(a_{j-1}, a_j)) \rceil$, since otherwise it would be $\lceil \mathcal{L}(a_j, a_{j+1}) \rceil \subset \lceil \mathcal{L}(a_{j+1}, a_{j+2}) \rceil \subset \lceil \mathbf{l}(\mathcal{L}(a_{j-1}, a_j)) \rceil \Rightarrow a_{j-1}, a_j \in \lceil \mathbf{l}(\mathcal{L}(a_{j-1}, a_j)) \rceil \subset \lceil \mathcal{L}(a_{j-1}, a_j) \rceil$. Therefore $a_{j+2} \in \lceil \mathbf{r}(\mathcal{L}(a_{j-1}, a_j)) \rceil$ that implies $\mathcal{L}(a_{j-1}, a_{j+2}) = \mathcal{L}(a_{j-1}, a_j)$

■

The reduction algorithm computes difference ($TRIG$) in $v = \mathcal{L}(a, b)$ and if it is not zero, returns v as a node which substitutes reduced a, b. Otherwise it returns NIL, i.e. nothing substitutes two reduced nodes.

ALGORITHM 2

```
function Reduce(a,b: node):node;
begin
   v = L(a,b);
   if TRIG(v,a,b)=1 then return(v)
   else return(NIL);
end;
```

When aplied to pair a_j, a_{j+1} the algorithm decides whether the pair to be deleted from $\mathbf{N}_\sigma$ completely, or it to be substituted by its LCA. If $TRIG(\mathcal{L}(a_j, a_{j+1}), a_j, a_{j+1}) = 1$, LCA substitutes a_j, and a_{j+1} to be deleted from $\mathbf{N}_\sigma$. By lemma 4.2(2,3), for node v it is true that: $a_{j-1} < v < a_{j+2}$, i.e. $\mathbf{N}'_\sigma = \mathbf{N}_\sigma \backslash \{a_j, a_{j+1}\} \cup v$ remains to be a set of mutually independent nodes, while the set $\mathfrak{M}_{\mathbf{N}'_\sigma} \subseteq \mathfrak{M}_{\mathbf{N}_\sigma}$, since $\mathcal{L}(a_{j-1}, v) = \mathcal{L}(a_{j-1}, a_j)$ and $\mathcal{L}(v, a_{j+2}) = \mathcal{L}(a_{j+1}, a_{j+2})$(Figure 4,b). Otherwise, i.e. $TRIG(\mathcal{L}(a_j, a_{j+1}), a_j, a_{j+1}) = 0$, pair a_j, a_{j+1} to be deleted from $\mathbf{N}_\sigma$ but nothing is added instead. New pair (a_{i-1}, a_{i+2}) emerges. However it does not lead to increasing number of $\mathfrak{M}_{\mathbf{N}'_\sigma}$ elements since(lemma 4.2(4)) $\mathcal{L}(a_{j-1}, a_{j+2}) = \max(\mathcal{L}(a_{j-1}, a_j), \mathcal{L}(a_{j+1}, a_{j+2}))$.

4.2 Generic algorithm of difference computation

Generic algorithm *DownUp* cuts leaves of the SD in order to find difference of an ELC and consists of two steps. At the first step $\mathbf{N}_\sigma$ is transformed to the state, when it is ordered in descending order of pairwise LCAs (i.e. $\lceil \mathcal{L}(a_{i-1}, a_i) \rceil \supset \lceil \mathcal{L}(a_i, a_{i+1}) \rceil$), reducing some nodes.

ALGORITHM 3

```
procedure DownUp;
ST:stack;
a,b,v:node;
with ST begin
//Part I - form the ordered stack
  push(N_σ[1]);
  for i = 2 to k do
    if [u.top] is empty then push(N_σ[i]);
    else if ⌈L([top],N_σ[i])⌉ ⊃ ⌈L([u.top],[top])⌉
    then begin
        repeat
              pop(a);
              pop(b);
              v=Reduce(a,b);
              if v ≠ NIL then push(v);
        until ([u.top] is empty) or
              (⌈L([top],N_σ[i])⌉ ⊂ ⌈L([u.top],[top])⌉)
        push(N_σ[i]);
    end;
    else push(N_σ[i]);
//Part II - reduce from the ordered stack
  while ([u.top] is not empty) do begin
    pop(a);
    pop(b);
    v=Reduce(a,b);
    if v ≠ NIL then push(v);
  end;
  if [top] is empty then λ = 0 else λ = 1
end.
```

Stack is used to store already processed ordered part of $\mathfrak{M}_{\mathbf{N}_\sigma}$. After all the array $\mathbf{N}_\sigma$ is processed, the

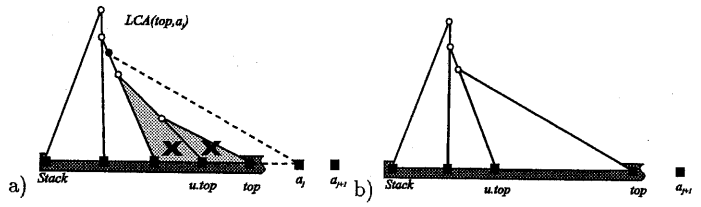

Figure 5: Reduction of stack

pairs to be exposed to the reduction procedure from the top of stack until it becomes empty. At this, pair on the top always complies with part (3) of the criteria. Initially, first pair a_1, a_2 is placed into the stack with two top elements accessible: ST[top] and ST[u.top]. On the $(i-2)$-th step of the computation current candidate on addiding to the stack is a_i. If $\lceil \mathcal{L}(ST[u.top], ST[top]) \rceil \supset \lceil \mathcal{L}(ST[top], a_i) \rceil$, then a_i is just added to the stack, otherwise as many nodes to be reduced from the top of stack that the latter condition becomes true. Clearly, reduced pairs satisfy to either part(1) or (2) of the criteria because they are minimal in the stack and less than $\lceil \mathcal{L}(ST[top], a_i) \rceil$. If MDAs have been precomputed for every node, on-line complexity of the algorithm could be estimated by a number of reductions that is $O(k_e) = O(k)$. On-line computation of k LCAs, according to [1], is completed in $kO(1) = O(k)$ at $O(N)$ preprocessing.

5 Root-to-bottom interpretation of LCs

The algorithm of down-up interpretation proposed above doesn't require preliminary construction of a signal diagram. Getting started from the set of independent event nodes and finding along the work their LCAs, the algorithm at the same time builds and traces the signal diagram. But computing time of the algorithm is $O(k)$ at any combination of input variables and pre-computed values of $v.F$ and $v.mda$ for each node (as it will be shown further).

The alternative approach of recursive up-down traverse requires preliminary construction of the signal diagram using 2 additional pointers $\mathbf{l}_{sd}, \mathbf{r}_{sd}$ which to be added to each node of the initial ELC. Then, difference in the root is computed as a recursive function of differences in its successors in the SD, and so on until the terminal nodes are reached, switch of which is equal to 1. It seems that all nodes of the SD to be passed if doing this way. However it is not necessary.

Consider node v of ELC and its children $\pi(v) = \{v.\mathbf{l}_{sd}, v.\mathbf{r}_{sd}\}$. Denote predicate $\mathfrak{R}_\lambda(w,v) = \mathfrak{R}(w,v) \wedge \partial(w)$. Clearly difference in $\lambda(v)$ can be derived recursively in a way simillar to Alg.3.2: $\partial(v) = (\mathfrak{R}_\lambda(v.\mathbf{l}_{sd}, \mathbf{l}(v)) \wedge \mathfrak{R}_\lambda(v.\mathbf{r}_{sd}, \mathbf{r}(v)) \wedge v.\varepsilon) \vee (\mathfrak{R}_\lambda(v.\mathbf{l}_{sd}, \mathbf{l}(v)) \wedge \overline{\mathfrak{R}_\lambda(v.\mathbf{r}_{sd}, \mathbf{r}(v))} \wedge v.\alpha_r) \vee (\mathfrak{R}_\lambda(v.\mathbf{l}_{sd}, \mathbf{l}(v)) \wedge \overline{\mathfrak{R}_\lambda(v.\mathbf{r}_{sd}, \mathbf{r}(v))} \wedge v.\alpha_r)$. As it is clear from the formula, recursive computation of λ in $v.\mathbf{l}_{sd}, v.\mathbf{r}_{sd}$ to be performed only if $\mathfrak{R}(v.\mathbf{l}_{sd}, v) = 1 (\mathfrak{R}(v.\mathbf{r}_{sd}, v) = 1)$. So that recursion in some nodes to be terminated and all the subtrees rooted in it to be excluded out of the part of SD to be processed.

Given an SD, let count the mean number of SD nodes to be passed. There two random factors that cause utility of probabilistic estimations: values of inputs and formula structure. Arbitrary choice of the structure means that a gate with equal possibility can be of **and** or **or** type. First, consider a $\mathbb{MIE}^2$ LC rooted in v with event nodes w_1, w_2. Clearly that probability of $\mathcal{D}(w_1) \supseteq \lceil v \rceil$ is dependent on distance between v and w_1. (The same, of course, is valid for w_2 too).

Let $p(v), q(v) = 1 - p(v)$ be the probabilities of that the input variable $\psi(v)$, associated with v, is in condition 1 or 0 respectively in state X. Consider path $[w_1 = u_0, u_2, \ldots, u_m = \mathbf{l}(v)]$. In a random $\mathbb{E}^1$ LCA, which $\lceil \mathbf{l}(v) \rceil$ certainly is, probability of that $\mathcal{D}(w_1) = u_{i+1}$ can be recursively expressed as $P_{i+1} = (\frac{1}{2}p(\eta(u_i)) + \frac{1}{2}(1 - p(\eta(u_i))))P_i = \frac{1}{2}P_i$. Thus P_i does not depend on probability distribution of input variable values and is equal $\frac{1}{2^m}$ if distance between event node w_1 and root of $\mathbb{MIE}^2$ circuit is $m+1$.

In a node v distances $d_l = |[v, v.\mathbf{l}_{sd}]|$, $d_r = |[v, v.\mathbf{r}_{sd}]|$ are used to find probabilities of $p_l = \mathbf{P}(\mathfrak{R}(v.\mathbf{l}_{sd}, v) = 1) = \frac{1}{2^{d_l}}$ and $p_r = \mathbf{P}(\mathfrak{R}(v.\mathbf{r}_{sd}, v) = 1) = \frac{1}{2^{d_r}}$). Let $e(v)$ be the mean value of the number of passed nodes in up-down trace of SD rooted in v. The following theorem connects value of $e(v)$ with corresponding characteristics of succeeding subtrees.

Theorem 5.1 $e(v) = 1 + p_l e(\mathbf{l}(v)) + p_r e(\mathbf{r}(v))$.

The theorem is a key to find estimation of the average number of nodes to be passed in the top-down recursive traverse at random input.

6 Application of preliminary inter-event computations and parallelization

Event-oriented nature of control systems enables to use time intervals between events to carry out certain preliminary computations (pre-computations) in order to accelerate response on coming event.

One of most important parameters, pre-computation of which brings essential benefit, is MDA of a node. For all nodes of LC it can be done in $O(N)$ steps. Having known values of MDA, the predicate $\mathfrak{R}(w,v)$ can be computed in constant time for an arbitrary pair of nodes (w,v). They also can be pre-computed for all the ancestors of the node at use of additional memory of $O(N \log N)$ size. Pre-computed values of markings: $v.\alpha_r$, $v.\alpha_l$, $v.\varepsilon$ reduce on-line computations in algorithm of node processing to check of 2-5 binary variables, providing total complexity of $\leq 5s$, where s is a number of steps in the algorithm. Amount of pre-computations is proportional to number of nodes in the LC i.e. is $O(N)$. Results illustrating influence of pre-computations on efficiency of algorithms are presented in the table 5.

	Alg. 4.2	Alg. 4.2 with precomputations	EDD traverse from [10]
On-line speed	$O(k \log N)$	$O(k)$	$O(k)$
Memory	$O(N)$	$O(N \log N)$	$O(N2^N)$
Pre-computations	-	$O(N)$	$O(N2^N)$

Figure 6: Comparison of EDD traverse and event computations in logic circuits

Parallelization also can be used to improve speed as in off-line as well as in on-line computations. It is possible to find all k LCAs simultaneously, as well as for every search use parallel algorithms from [1], [3]. Besides, it is possible to process nodes of each layer $\min(\mathfrak{M}_{\mathbf{N}_\sigma})$, which are candidates to be reduced, independently and, hence, simultaneously.

7 Conclusion

New approach to event-driven Boolean computations has been proposed in the paper. It uses presentation of the computed formula as logic circuit that is more compact than binary decision diagrams known as those can be processed of most speed. It has been proven that in event-driven computations proposed data structure can be traversed as fast as the BDDs. However, memory and that is most important recomputation demand of the circuits in order to provide equally fast on-line responce is conciderably lower (required memory depends on number of input variables linearly rather than exponentially as it is shown in table in Fig.5).

Contribution of the paper is in the following:

- Problem of event-driven computation has been solved for the general case of arbitrary number of triggered inputs and arbitrary number of entries of each input into the formula.
- It has been proven that algorithm of down - up traverse of logic circuits provides on-line responce in the same order of time as the best event-oriented algorithms using BDDs do at much more expences of memory and precomputation.
- It has been shown that straighforward recursive top-down interpretation of the logic circuits also can be beneficial in a view of better probabilistic estimations.
- Both the down-up and top-down algorithms can benefit on parallel computations, that opens the way of effective application of parallel and distributed architectures in the logic control.

However, the developed algorithms inherit major drawbacks of the event-driven computations, the most serious of those is that they do not improve the worst case since some precomputations are always required. Nevertheless they can be applied under assumption about reasonable interval between subsequent events. As for the continuation of this work, since the data structure that unify logic circuits and decision diagrams has been proposed recently in [4], we plan to generalize our approaches to event-driven computation of the both into the homogeneous algorithm.

References

[1] Sheiber B. and Vishkin U. On finding lowest common ancestors: Simplification and parallelization. *SIAM J.Comput.*, 17(6):1253–1262, 1988.

[2] C.Y.Lee. Representation of switching circuits by binary-decision programs. *Bell. Syst. Tech. J*, 6:985–999, 1959.

[3] E.Schenk. Parallel dynamic lowest common ancestors. *SWAT '94. 4th Scandinavian Workshop on Algorithm Theory. Proceedings*, pages 302–13, 1994.

[4] Andersen H.R. and Hulgaard H. Boolean expression diagrams. In *Proc. of LICS'97*, 1997.

[5] Davio M., J.-P.Dechamps, and A.Thayse. *Discrete and Switching Functions.* McGraw-Hill, New York, 1978.

[6] Akers S.B. On a theory of boolean functions. *SIAM Journal*, 7(4):487–498, 1959.

[7] John T.Welch. The clause counter map: An event chaining algorithm for online programmable logic. *IEEE Trans. on Robotics and Automation*, 2, 1995.

[8] V.Viatkin, K.Nakano, and T.Hayashi. Evaluation of logic expressions based on event-oriented interpretation of marked functional diagrams. In *Proc. of 35th international conference of Society for instrumentation and control engineering of Japan*, pages 1243–1248. SICE, 1996.

[9] V.Viatkin, K.Nakano, and T.Hayashi. Logic evaluations as processing of queries using binary decision diagrams. *IPSJ SIG Notes*, 96-89:31–38, 1996.

[10] V.Viatkin, K.Nakano, and T.Hayashi. Optimized processing of complex events in discrete control systems using binary decision diagrams. In *Proc. of International workshop on algorithms and architectures in real-time control*, pages 445–450. IFAC, 1997.

[11] V.Viatkin, N.Ishii, and T.Hayashi. Event oriented evaluations of binary decision diagrams. In *Proc. of International workshop on discrete event systems*, pages 374–379, Edinburgh, 1996. IEE.